School of Engineering and Applied Science (SEAS), Ahmedabad University

CSE400: Fundamentals of Probability in Computing

Problem Statement: Constrained Vehicle Routing for Minimizing Time, Distance, and Budget Costs

Group Name: s1-its-3
Team Members:

- Prina Patel (AU2340040)
- Priyanka Kapoor (AU2340093)
- Purvish Parekh (AU2340128)
- Pratham Sandesara (AU2340239)
- Misha Bajaj (AU2340265)

I. Background and Motivation

- **Background**
  The main aim of the Intelligent Transportation System (ITS) is to enhance mobility, improve the efficiency of transportation, and reduce traffic congestion using computational methods. As urban areas are expanding at higher rates, transportation efficiency has become a critical issue. The main problem with traditional routing algorithms is that they only focus on the shortest path and ignore real-world constraints like traffic congestion, budget limitations, and variable travel time. This project will address these challenges by integrating probabilistic models, optimization techniques and real-time adaptability.

  So, to overcome the limitations of traditional algorithms, which ignore real-world constraints, we have used the A* algorithm approach. To handle uncertainties, travel time, distance, traffic congestion, and cost are taken as random variables. By this, real-world uncertainties can be handled in a better way.

- **Motivation**
  The motivation behind this project is to solve current transportation system problems and inefficiencies because these problems create transportation delays, unnecessary costs, and fuel usage. Traditional routing methods generally ignore the uncertainties of the real world, due to which inefficient route choices may be made. So, by using computational models and simulations, we aim to generate smarter and more accurate routing decisions.

  1. **Reducing Travel Costs and Time:**
     Efficient vehicle routing has a direct impact on fuel consumption, delivery schedules, and logistics efficiency. By developing an algorithm that considers constraints such as time, distance, and budget, this project aims to improve operational efficiency in urban mobility.
  2. **Smarter Route Planning:**
     In normal or traditional route planning, generally, the routes are selected mostly on basis of minimum time or shortest distance. In this the unexpected factors like delay due to traffic congestion, fuel prices, risk of vehicle breakdown, etc. are ignored. Our system uses all required data and finds out the better routes, which saves time and money[1].

II. Application

(a) Route Optimization

(b) Time Optimization

1. **Delivery Services:**
   To handle the delivery of various users, companies like Flipkart, Swiggy, Amazon, Uber, etc., depend on optimized routing. So, routes must be planned in such a way that helps to minimize distance, time, and cost to ensure on-time deliveries.

2. **Healthcare and Medical Supply Delivery:**
   In the medical domain, vehicle routing plays a crucial role because timely deliveries of lab samples, medical supplies, or any other emergency situation are important. Thus, minimizing time and distance and optimizing routing will help to reduce the overall cost.

3. **Fleet Management Systems:**
   To manage the traffic effectively, fleet management systems generally depend on efficient routing algorithms. By collecting the data on traffic, weather, and based on priorities, these technologies provide the best route to drivers, which ensures on-time deliveries.

4. **Public Transportation:**
   Routing algorithms are used to plan the routes of buses. It is also used for scheduling the time in such a way that minimizes waiting time, avoids the routes with traffic, and also reduces the travel distance. Real-time GPS data allows the companies to adjust the routes in such a way that helps to minimize the time.

III. T1 - Mathematical Modelling and Mathematical Analysis

The following random variables are used:

1. Travel Time:

Type: Continuous Random Variable

Probability Distribution: Normal Distribution (PDF)

$$f_T(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(t-\mu)^2}{2\sigma^2}}, \quad -\infty < t < \infty$$

Where, μ is the average travel time and σ is the standard deviation

2. Fuel Cost:

Type: Continuous Random Variable

Probability Distribution: Exponential Distribution (PDF)

$$f(F) = \lambda e^{-\lambda F}, \quad F \geq 0$$

Where λ is the rate parameter and it is inversely related to the average cost

3. Traffic Congestion:

Type: Continuous Random Variable

Probability Distribution: Normal Distribution (PDF)

$$f_C(C_t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(C_t-\mu_C)^2}{2\sigma_C^2}}, \quad -\infty < C_t < \infty$$

4. Distance Travelled:

Type: Continuous Random Variable

Probability Distribution: Normal Distribution (PDF)

$$f_D(d) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(d-\mu)^2}{2\sigma^2}}, \quad d \geq 0$$

5.Vehicle Breakdown:

Type: Discrete Random Variable

Probability Distribution: Bernoulli Distribution(PMF)

$$P(B = b) = \begin{cases} p, & \text{if } B = 1 \text{ (Breakdown occurs)} \\ 1 - p, & \text{if } B = 0 \text{ (No breakdown)} \end{cases}$$

where p is the probability of a vehicle breakdown occurring.

- **Travel Time** -

  Now, for calculating travel time, the following summation equation will be used:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} T_{ij} X_{ij}$$

Double summation used in this equation represents all possible travel routes (from location i to location j) for N nodes.

T_{ij} is the travel time between i and j.

X_{ij} This is the decision variable. It indicates if a particular route (i to j) is chosen or not. It can be represented as:
$X_{ij} = 1$: If the route is selected
$X_{ij} = 0$: If the route is not selected

In this equation, as travel time is taken continuous, we will now take the expectation of total travel time, which is then calculated using integration over its PDF. And we want to minimize the total time so equation will be:

$$min \sum_{i=1}^{N} \sum_{j=1}^{N} E[T_{ij}] X_{ij}$$

$E[T_{ij}]$ will be calculated in this way:

$$E[T_{ij}] = \int_0^\infty t f_{T_{ij}}(t)\, dt$$

- **Travel Distance** -
  Now, for calculating distance, the following summation equation will be used:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} D_{ij} X_{ij}$$

  Here, D_{ij} is the travel distance between node i and j.

  In this equation, as distance is taken continuous, we will now take the expectation of total distance, which is then calculated using integration over its PDF. We want minimum distance so equation will be:

$$min \sum_{i=1}^{N} \sum_{j=1}^{N} E[D_{ij}] X_{ij}$$

  $E[D_{ij}]$ will be calculated in this way:

$$E[D_{ij}] = \int_0^\infty d f_D(d)\, dd$$

- **Total Cost** -
  Now, for calculating total cost, the following summation equation will be used:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} C_{ij} X_{ij}$$

Here, C_{ij} represents the cost of traveling from i to j.

In this equation, as cost is taken continuous, we will now take the expectation of total cost, which is then calculated using integration over its PDF. For minimum total cost equation will be:

$$min \sum_{i=1}^{N} \sum_{j=1}^{N} E[C_{ij}] X_{ij}$$

$E[C_{ij}]$ will be calculated in this way:

$$E[C_{ij}] = \int_0^\infty c \, f_{C_{ij}}(c) \, dc$$

Joint Probability Model

Now, to find the minimum time, distance, and cost we will use the N variate (trivariate) joint probability model in following way:

$$\begin{aligned} P(T, D, C) &= P(T)P(D|T)P(C|T, D) \\ &= f_T(t) \cdot f_{D|T}(d|t) \cdot f_{C|T,D}(c|t, d) \end{aligned}$$

We will use respective probability density functions of each term so equation will be:

$$P(T, D, C) = \left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \right) \cdot \left(\frac{1}{\sigma_d\sqrt{2\pi}} e^{-\frac{(d-(\alpha t+\beta))^2}{2\sigma_d^2}} \right) \cdot \left(\lambda e^{-\lambda c} \right)$$

Where, $\alpha t + \beta$ is the expected distance for a given travel time t

- $P(T)$: It is the marginal probability of travel time T
- $P(D \mid T)$: It is the conditional probability of travel distance D given a specific travel time T. This shows that distance and time are interconnected (i.e. longer distances take more time).
- $P(C \mid T, D)$: It is the conditional probability of travel cost C given both travel time T and travel distance D.
- $f_T(t)$: It is the PDF of travel time T, which represents probability of different travel times.
- $f_{D|T}(d \mid t)$: It is the conditional PDF of travel distance D given travel time T which shows how distance varies depending on the given time.
- $f_{C|T,D}(c \mid t, d)$: The conditional PDF of travel cost C given travel time T and travel distance D. It shows how cost is distributed depending on both time distance.[2][3][4][5]

IV. T2 - Code (with description of each line)

A. Importing Required Libraries

```
1  from random import uniform, shuffle
2  import sumolib
3  import networkx as nx
4  import matplotlib.pyplot as plt
```

Figure 1: Importing Libraries

Here, the following libraries are imported as required:

1. random: To generate random numbers and add randomization in code
2. sumolib: To load and use the SUMO network file
3. netwrokx: To generate and plot the graph
4. matplotlib.pyplot: To plot the graphs

B. Loading SUMO Network and Generating Graph

```
6  net = sumolib.net.readNet('csr.net.xml')
7  G = nx.DiGraph()
8  node_positions = {}
9
10 for node in net.getNodes():
11     node_id = node.getID()
12     G.add_node(node_id)
13     node_positions[node_id] = node.getCoord()
14
15 edge_map = {}
16 for edge in net.getEdges():
17     from_node = edge.getFromNode().getID()
18     to_node = edge.getToNode().getID()
19     edge_id = edge.getID()
20     weight = edge.getLength()
21
22     G.add_edge(from_node, to_node, weight=weight)
23     edge_map[(from_node, to_node)] = edge_id
```

Figure 2: Generating graph from SUMO network file

Here, the following lines reads the SUMO network file and generates a graph.

1. Reading SUMO network file: Lines 6-8 reads the 'csr.net.xml' file using the 'sumolib', and creates a graph using 'networkx'
2. Adding Nodes to the Graph: Lines 10-13 Loop over every node from the network file and add it to graph G and store their coordinates.

3. Adding directed edges to Graph: Lines 15-23 Loop over every edge from the network, take from and to nodes for the edge, assign distance as the weight of the edge, and add it to the graph.

C. Heuristic Function

```
25  def heuristic_randomised(n):
26      x1, y1 = node_positions[n]
27      x2, y2 = node_positions[stop_node]
28      base_distance = ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5
29      noise = uniform(-0.07, 0.07) * base_distance
30      return base_distance + noise
31
32  def get_neighbors_randomized(v):
33      if v in G:
34          neighbors = [(neighbor, G[v][neighbor]['weight']) for neighbor in G[v]]
35      else:
36          neighbors = []
37      shuffle(neighbors)
38      return neighbors
```

Figure 3: Heuristic Function

4. Heuristic Function: Lines 25-30 define the Randomized Heuristic function, which returns the distance between the start node and the stop node passed as the parameter. Line 29 adds additional noise in the function depicting the traffic.

5. Neighbor Function: Lines 32-38 define the Randomized Get Neighbor function, which returns the list of neighbors of that given node if any. Additional line 37 shuffles the neighbors to explore different paths.

D. A* Algorithm

This function defines the A* Algorithm[6].

1. Initialization of function: Lines 41-44 initialize the function parameters such as 'open-set' and 'closed-set', which denote not explored and explored nodes, respectively. It also initializes cost and parent dictionaries.

2. Iterating till stop-node: Lines 46-72 run a while loop which continues till the destination node is reached or no node can be found.
   Line 47 chooses the node with the minimum estimated cost comprising the cost, randomized heuristic, and minor noise. Line 49 checks if the stop-node is reached, and if reached, lines 50-56 compute the path and return path edges.
   Lines 58-59 remove the current node from the open-set and add it to the closed-set.
   Lines 61-72 iterate over all the neighbors of the current node. Calculate the estimated cost and choose the efficient node. If the neighbor node is already explored, then skip the node. Lines 74-75 return None if no path exists.

```
40  def astaralgo_randomised(start_node, stop_node):
41      open_set = set([start_node])
42      closed_set = set()
43      cost = {start_node: 0}
44      parents = {start_node: None}
45
46      while open_set:
47          n = min(open_set, key=lambda node: cost[node] + heuristic_randomised(node) + uniform(0, 1e-3))
48
49          if n == stop_node:
50              path_edges = []
51              while parents[n] is not None:
52                  path_edges.append(edge_map[(parents[n], n)])
53                  n = parents[n]
54              path_edges.reverse()
55              print("Path found: {}".format(path_edges))
56              return path_edges
57
58          open_set.remove(n)
59          closed_set.add(n)
60
61          for m, weight in get_neighbors_randomized(n):
62              if m in closed_set:
63                  continue
64
65              tentative_cost = cost[n] + weight
66              if m not in open_set:
67                  open_set.add(m)
68                  parents[m] = n
69                  cost[m] = tentative_cost
70              elif tentative_cost < cost[m]:
71                  cost[m] = tentative_cost
72                  parents[m] = n
73
74      print("Path does not exist!")
75      return None
```

Figure 4: A* Algorithm

E. Initializing Nodes

```
77  start_node = '1779066710'
78  stop_node = '3779336540'
79  path_edges = astaralgo_randomised(start_node, stop_node)
```

Figure 5: Initializing Nodes

Lines 77 and 78 initialize the start-node and stop-node respectively. Line 79 calls the "astaralgo" function for the start and stop-node and takes path-edges.

F. Writing to SUMO xml file

Lines 81-93 define the function to write the path-edges passed as a parameter. It checks if the path is not empty and if not, it opens the 'test.rou.xml' file write mode and writes the

```
81  def write_routes(path_edges):
82      if path_edges is None:
83          print("No valid path found. Cannot generate route file.")
84          return
85
86      with open("test.rou.xml", "w") as f:
87          f.write("""<routes>
88      <vType id="car" accel="1.0" decel="5.0" sigma="0.5" length="5" maxSpeed="50"/>
89      <vehicle id="1000" type="car" depart="1" color="1,0,0">
90      <route edges="{}"/></vehicle>
91  </routes>""".format(" ".join(path_edges)))
92
93      print("Route file `test.rou.xml` generated.")
94
95  write_routes(path_edges)
```

Figure 6: Writing to SUMO xml file

details required by SUMO. Line 95 calls the following function in path-edges.

G. Visualising the Graph

```
97  plt.figure(figsize=(15, 25))
98  nx.draw(G, pos=node_positions, with_labels=True, node_color='lightblue', edge_color='gray',
99          node_size=250, font_size=4, arrows=True, connectionstyle="arc3, rad=0.02")
100
101 if path_edges:
102     path_nodes = []
103     for (from_node, to_node), edge_id in edge_map.items():
104         if edge_id in path_edges:
105             path_nodes.append((from_node, to_node))
106
107     nx.draw_networkx_edges(G, pos=node_positions, edgelist=path_nodes, edge_color='green', width=2)
108
109 plt.title("SUMO Network Visualization with Highlighted Path")
110 plt.show()
```

Figure 7: Ploting the Graph with Highlighted Path

Lines 97-110 plot the graph using 'networkx' and 'matplotlib' libraries.
Lines 97-99 plot the graph and assign the value for parameters such as node color, edge color, node size, font size, etc, as shown in the image. Lines 101-107 check if resultant path-edges exist, and if yes, change the edge color to green to highlight the efficient path. Lines 109-110 put the title of the graph and show the graph plot.

H. SUMO Configuration File

Here is the SUMO configuration file used in our project. This file is used to instruct SUMO to use which file.
Lines 8-12 specify input for the SUMO. Line 9 shows the net file 'csr.net.xml' which is the SUMO network file. Line 10 shows the route files 'test.rou.xml', which contain the path

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- generated on 2024-06-12 18:31:58 by Eclipse SUMO sumo Version v1_20_0+0443-233e48e5c88
-->

<sumoConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/sumoConfigurati

    <input>
        <net-file value="csr.net.xml"/>
    <route-files value="test.rou.xml,trips.trips.xml"/>
    <additional-files value="map.poly.xml"/>
    </input>

    <time>
        <begin value="0"/>
        <end value="10000"/>
    </time>


</sumoConfiguration>
```

Figure 8: SUMO Configuration File

generated by the A* algorithm, and 'trips.trips.xml', which contain data for the random traffic. Line 11 shows the additional file to load background data for the simulation. Lines 14-17 specify the run time for the simulation.

V. T4 - Algorithm(Deterministic/Baseline and Randomized)

A. Deterministic A* Algorithm

Algorithm 1 Simple A* Algorithm

Input: Start node S_0, goal node S_g, heuristic $h(n)$, cost $g(n)$
Open list $\leftarrow S_0$, Closed list $\leftarrow$ empty
while Open list is not empty **do**
    Pick node n in Open list with smallest $f(n) = g(n) + h(n)$
    if $n = S_g$ **then**
        return path from S_0 to S_g
    end if
    Move n from Open to Closed list
    for each neighbor m of n **do**
        if m is in Closed list **then**
            Skip m
        end if
        $g(m) = g(n) + \text{cost}(n, m)$
        if m is not in Open list OR new $g(m)$ is better **then**
            Set parent of m as n
            $f(m) = g(m) + h(m)$
            Add or update m in Open list
        end if
    end for
end while
return No path found

The A* algorithm is a widely used pathfinding and graph traversal algorithm that guarantees finding the shortest path in a weighted graph.

Limitations of the Deterministic A* Algorithm
While the deterministic A* algorithm efficiently finds the optimal path in static environments, it faces several limitations in real-world applications:

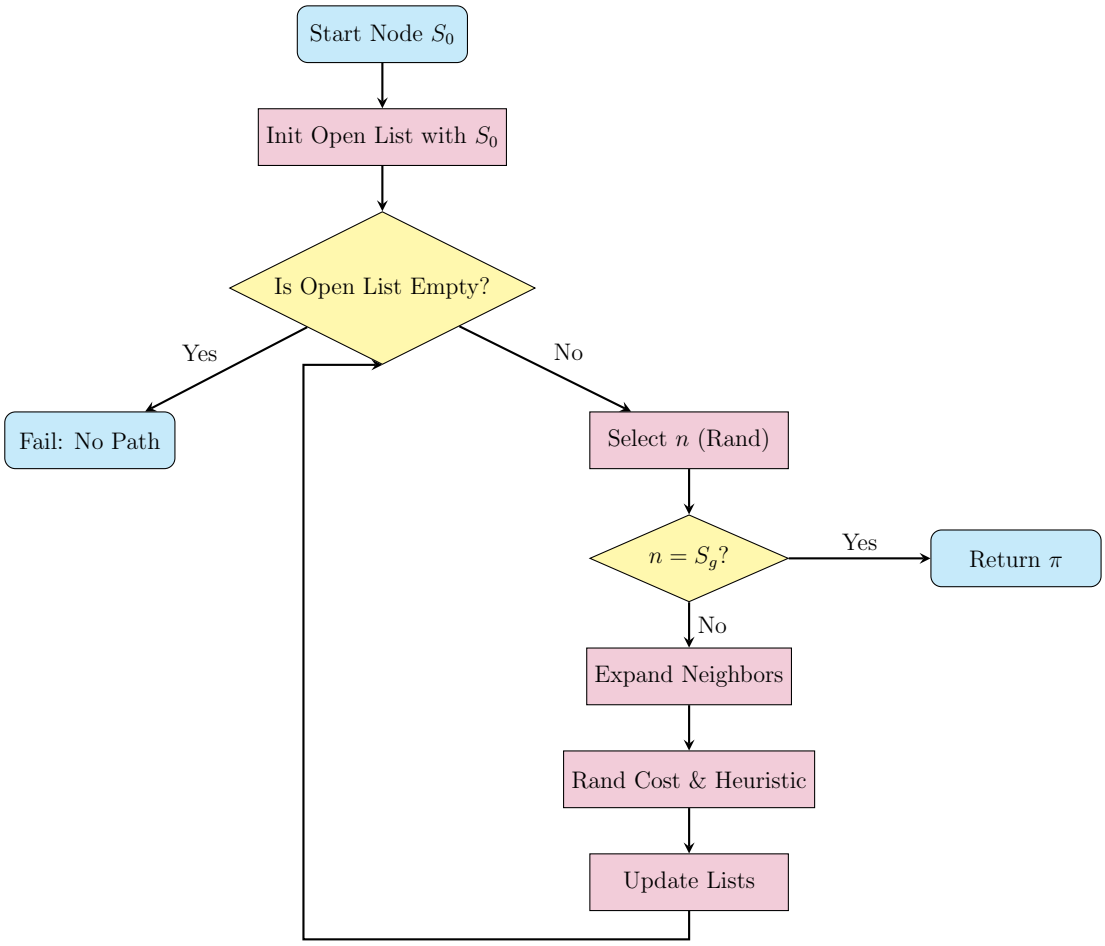
- **Fixed Costs**
- **Strict Node Selection**
- **Inflexibility in Dynamic Environments**
- **Lack of Exploration**

B. Randomized A* Algorithm

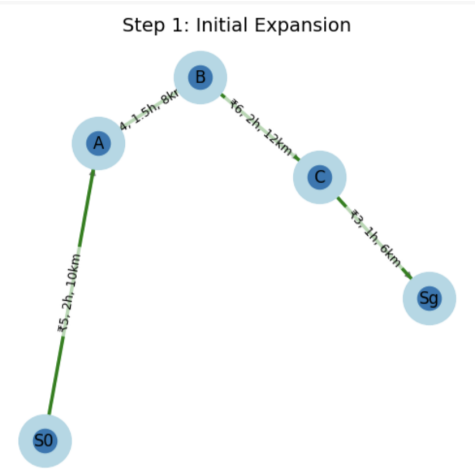
Algorithm 2 Randomized A* Algorithm

Input: Start state S_0, goal state S_g, heuristic $h(n)$, cost function $g(n)$
Initialize open set *Open* with S_0, closed set *Closed*, and path π
while Open is not empty **do**
    Select node n from Open with $f(n) = g(n) + h(n)$
    if Randomized Node Expansion **then**
        With probability p, expand node with minimum $f(n)$
        With probability $1 - p$, expand a random neighbor node
    end if
    Remove n from Open, add to Closed
    if n is goal S_g **then**
        Return path π
    end if
    for each neighbor m of n **do**
        if m is not in Closed **then**
            Update cost $g(m) = g(n) + \text{cost}(n, m)$
            if Randomized Cost Function **then**
                $w_1 \sim \mathcal{N}(\mu_T, \sigma_T^2)$ for travel time
                $w_2 \sim \text{Exp}(\lambda)$ for fuel cost
                $w_3 \sim \text{Bern}(p)$ for breakdown probability
                Add random perturbation to $g(m)$
            end if
            if m is not in Open or new path is better **then**
                Set parent of m to n
                if Randomized Heuristic Function **then**
                $h(m) = w_1 \cdot T^*(m) + w_2 \cdot F^*(m) + w_3 \cdot B^*(m) + \epsilon$
                where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ (Gaussian noise)
                end if
                Add m to Open
            end if
        end if
    end for
end while
Return NO PATH FOUND

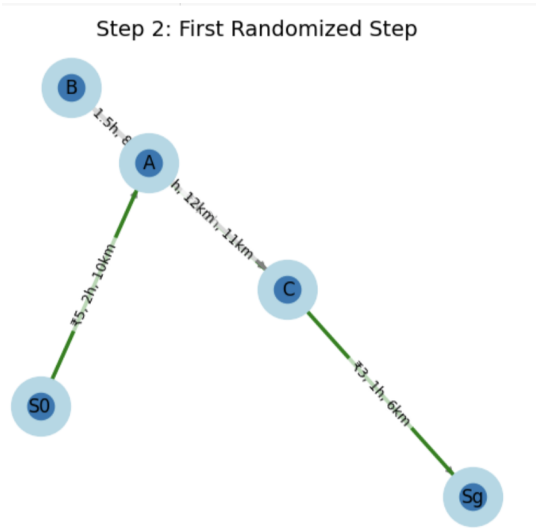
C. Flowchart of Randomized A* Algorithm



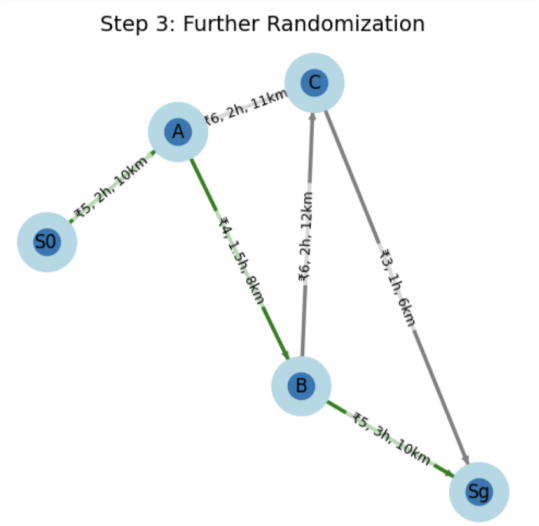
D. Step By Step Randomization Process



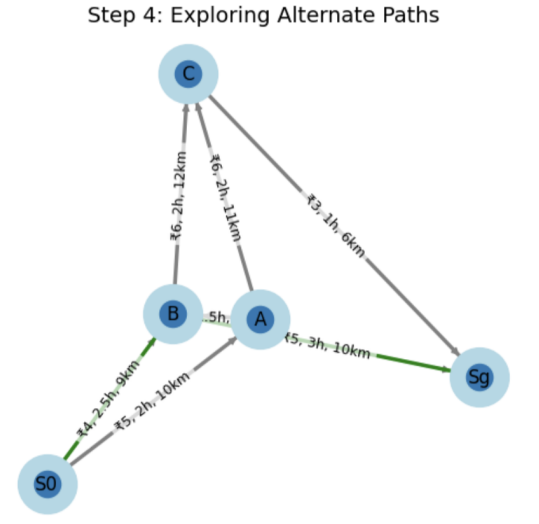
- The algorithm begins its operation from the start node S_0.
- It identifies the neighboring node and expands to node A.
- From node A, it proceeds to node B following a deterministic greedy policy.
- No randomization is involved at this point—decisions are fully based on path cost.
- This step sets up the initial structure for further exploration.



- Now Randomization is introduced in pathfinding.
- Instead of deterministically choosing the shortest path from A, the algorithm chooses node C.
- This introduces variability and enhances exploration in unknown terrain.
- From C, the algorithm advances directly to the goal node S_g.
- The path is suboptimal in cost but covers different node regions.



- Increased randomness influences the path decision.
- From A, instead of C, the path goes back to B—despite already visiting it.
- From B, the algorithm moves forward to C before proceeding to the goal.
- This creates an unnecessary loop, representing a less efficient trajectory.
- Demonstrates algorithm's non-greedy behavior for broader exploration.



- The algorithm re-evaluates earlier decisions to test alternate paths.
- It now tries the sequence: $S_0 \rightarrow B \rightarrow A \rightarrow C \rightarrow S_g$.
- This helps discover equivalent or better-cost paths not seen earlier.
- Encourages exploration beyond local optimality to find globally efficient routes.
- Useful in dynamic or changing environments with shifting costs.

- The final path is selected using experience and randomized evaluation.
- It balances exploration (unvisited nodes) and exploitation (shortest cost)
- The final path might not be the shortest, but it helps explore more parts of the network.
- It works well even when the environment is uncertain or not fully known.
- It shows how random choices and smart guesses (heuristics) can work together effectively.

E. Explaination :

The algorithm maintains two main lists:

- **Open List:** Contains nodes that need to be checked.
- **Closed List:** Contains nodes that have already been checked.

The algorithm follows these key steps:

1. Initialize the open list with the start node and set its cost to zero.
2. While the open list is not empty:
   (a) Select a node from the open list based on a randomized expansion strategy.
   (b) If the selected node is the goal, return the reconstructed path.
   (c) Otherwise, expand the selected node and update its neighbors.
   (d) Add the selected node to the closed list.
3. If no path is found, return failure.

Randomized Node Expansion

- Unlike the traditional A* algorithm, which always selects the node with the lowest cost function $f(n) = g(n) + h(n)$, the randomized approach introduces probability into node selection:

- With probability p, select the best node (smallest $f(n)$).
- With probability $1 - p$, select a random neighboring node.

Randomized Cost Function

- The cost function $g(n)$ is slightly changed randomly to reflect real-world uncertainties, such as unpredictable conditions.

- **Travel Time:** Sampled from a normal distribution $\mathcal{N}(\mu_T, \sigma_T^2)$.
- **Fuel Cost:** Sampled from an exponential distribution $\text{Exp}(\lambda)$.
- **Breakdown Probability:** Sampled from a Bernoulli distribution $\text{Bern}(p)$.

Randomized Heuristic Function

- The heuristic function $h(n)$ is also adjusted by adding a small random value ϵ to account for uncertainty.

$$h(n) = w_1 \cdot T^*(n) + w_2 \cdot F^*(n) + w_3 \cdot B^*(n) + \epsilon,$$

Concept and Mechanism

- The traditional A* algorithm finds the shortest path by evaluating nodes based on the function:

$$f(n) = g(n) + h(n)$$

- $g(n)$ is the cost from the start node to n.
- $h(n)$ is the heuristic estimate of the cost from n to the goal.

- Randomized A* modifies this approach by introducing a small random factor $r(n)$ to the heuristic:

$$f'(n) = g(n) + h(n) + r(n)$$

where $r(n)$ is typically a small value drawn from a uniform or Gaussian distribution.

VI. T3 - Results and Inferences (Domain + CS perspective)

A. CS perspective

Efficient vehicle routing is critical for minimizing travel time, distance and operational costs and this task is increasingly complex in dynamic urban environments where conditions like congestion, road closure and signal delays must be considered. We will compare two algorithm approaches and we will evaluate through Computer Science perspective.

- **Time Complexity**

	A* Algorithm	Randomized A* Algorithm
Time Complexity	$O(b^d)$	$O(NlogN)$

  - The worst time complexity of A* algorithm of $O(b^d)$ where
    - b is the Branching factor(number of possible moves from a node).
    - d is the depth of the optimal solution.
  - The average time of Randomized A* algorithm is $O(NlogN)$ where N represents number of Nodes(roads intersection).
  - Randomized A* algorithm offer more efficient scaling and execution in real-time environments.

- **Graph Size**

	A* Algorithm	Randomized A* Algorithm
Graph Size	Small(10x10 grids)	Large(thousands of roads and intersections).

  - A* algorithm is only applicable for 10×10 grid and it is static with fixed edge weight.
  - In Randomized A* algorithm, dynamic and large graphs are handled efficiently due to SUMO's internal routing logic and optimized graph structure.

- **Traffic Modeling**

	A* Algorithm	Randomized A* Algorithm
Traffic Modeling	No traffic consideration	Includes congestion, delays and signals.

  - A* algorithm operates on a static graph where edge weight do not change.
  - In A* algorithm each route calculated is optimal only at time of execution and doesn't respond to real-world dynamics.
  - Randomized A* algorithm edges have dynamic weights, and routing adapts in real-time based on live simulation conditions.

- **Real-time Adaptability**

	A* Algorithm	Randomized A* Algorithm
Real-time adaptability	Static	Dynamic, real time

  - In A* Algorithm when route is computed then it does not change.Any change in environment requires re-running the algorithm.
  - Randomized A* Algorithm updates in real-time with the involvement of traffic condition.

- **Delivery/Logistics**

	A* Algorithm	Randomized A* Algorithm
Delivery/Logistics	Finds shortest path	Optimizes based on real-world conditions.

  - The standard A* algorithm uses heuristics to choose the shortest path
  - Randomized A* algorithm controls the randomness in path selection.
  - Randomized A* algorithm is best suited as its efficiency in real-world conditions matter more than just minimizing distance.

A* algorithm is efficient only for small, deterministic environment but it fails to adapt or scale for large real-world network and on the other hand Randomized A* algorithm are scalable, efficient and adaptive. Randomized A* algorithm is well suited for intelligent transport applications where real-time decision making and dynamic routing are essential.

B. Project Domain

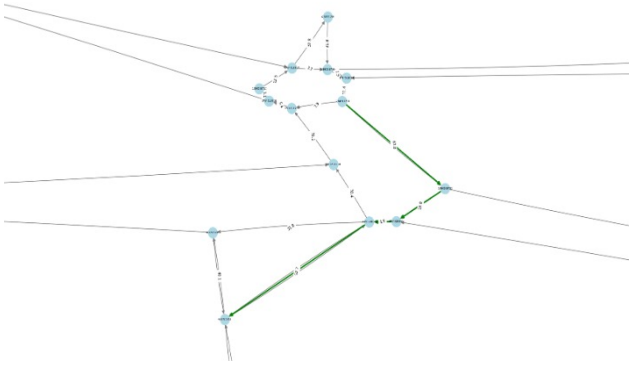


Figure 9: Representation of road-network

Figure-9 shows a graph based-representation of a road network where:

- Nodes: It represents road points.
- Edges: It represents roads between junction.

- Edge Weight: It indicates cost value.
- Green highlighted path: It represents the route chosen by the Randomized A* algorithm.

The visualization demonstrates how the Randomized A* Algorithm works on predefined network helping in exploring alternative routes that may avoid congestion or high-cost paths.

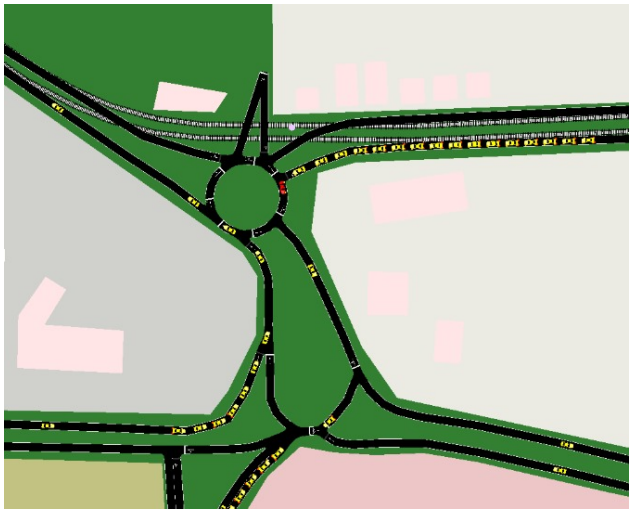
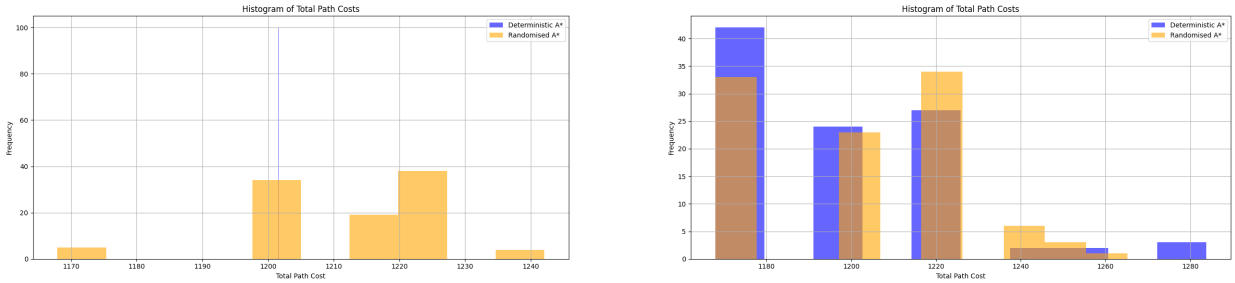


Figure 10: Real time traffic simulation

Figure-10 demonstrates real-world traffic conditions. A* algorithm would choose a shortest path, even if it's congested, but Randomized A* algorithm choose alternate path which would eventually reduce waiting time and fuel consumption. Randomized A* algorithm would help to improve traffic flow by distributing traffic evenly. This would also help in making route adaptive and ideal for real-time decision making.



(a) Cost Histogram Before Traffic

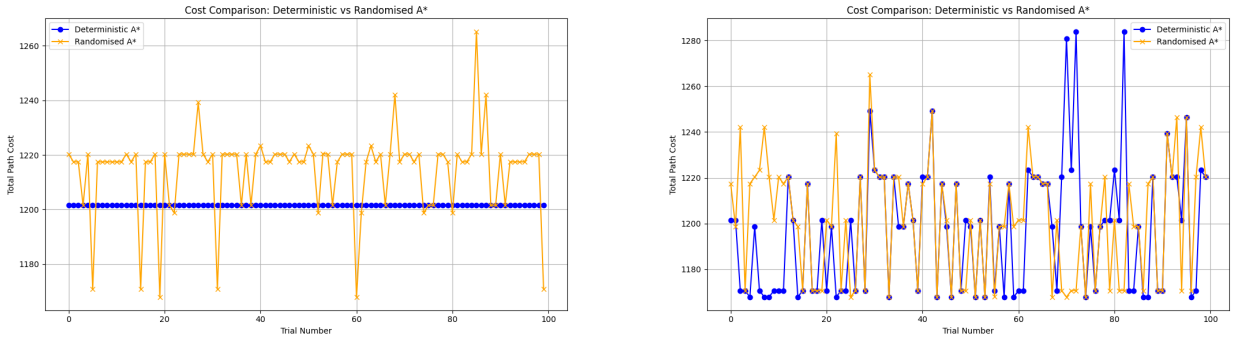
(b) Cost Histogram Before Traffic

Figure 11: Plots for Cost Histogram

Figure-11 demonstrates Cost Histogram Before and After Traffic.

- X-axis is the total path cost

- Y-axis is the frequency of how many times that cost occurred in trials
- Before Traffic
  - Deterministic A* has a sharp peak around a single cost.
  - Randomized A* has more spread out but not too extreme.
- After Traffic
  - Deterministic A* is more concentrated.
  - Randomized A* has a wider spread, still having a long tail.
  - Though Randomized A* is less reliable, it finds lower cost path occasionally.



(a) Cost Comparision Before Traffic

(b) Cost Comparision After Traffic

Figure 12: Plots for .....

Figure-12 demonstrates Cost Comparision Before and After Traffic.

- X-axis shows trial numbers
- Y-axis shows total path costs
- Before Traffic
  - Deterministic A* produces a consistent costs which tells us that it follows the same logic.
  - Randomized A* has a variable cost and this fluctuates due to randomness in path selection.
- After Traffic
  - Deterministic A* has more variability than before as traffic data is making deterministic path calculation more sensitive
  - Randomized A* varies widely but its average range overlaps with deterministic one.

VII. T5 - Derivation of Bounds and Results (new inferences)

A. Markov's Inequality

The upper limit for probability of surpassing a threshold value is determined through Markov's Inequality. The inequality operates on non-negative random variables through their mean values. The forecast and management of extreme travel delays can be conducted without requiring complete traffic data using this approach.

Bound:

$$P(T \geq \alpha) \leq \frac{E[T]}{\alpha}$$

- T = Travel time,
- $E[T]$ = Expected travel time,
- α = Delay threshold.

Application to CVRP:

1. Helps prevent worst-case delays by ensuring travel time rarely doubles.
2. Improves route reliability even with traffic changes.
3. Rejects routes where delay probability exceeds set limits.
4. Adjusts delay thresholds dynamically using real-time data.

B. Chebyshev's Inequality

This inequality estimates how likely any value can deviate from the mean. To check this it uses both mean and variance. This inequality is useful for predicting the travel delays caused due to traffic. We can use this inequality to handle the uncertainty in travel time caused due to traffic congestion.

Bound:

$$P(|T - \mu| \geq k\sigma) \leq \frac{1}{k^2}$$

Where:

- μ = Mean travel time,
- σ = Standard deviation,

- k = Number of standard deviations from the mean.

Application to CVRP:

1. It helps in delay management, thus ensuring on-time deliveries.
2. It also helps to analyse the traffic by prediciting how travel time will deviate from average, which helps to avoid the unexpected delays.

C. Cauchy-Schwarz Inequality

This inequality creates a relationship between any two random variables, and it also helps link their joint behavior to their individual variations. So, it helps to identify how closely related two factors are (for example travel time and fuel cost) by bounding their correlation.

Bound:

$$|E[TF]| \leq \sqrt{E[T^2] \cdot E[F^2]}$$

Where:

- T = Travel time
- F = Fuel cost
- $E[\cdot]$ = Expectation operator

Application to CVRP:

1. It helps to measure the dependence between random variable like travel time and fuel cost
2. It helps to guide fuel cost decisions by bounding combined uncertainty

References

[1] C. Wang, Z. Cao, Y. Wu, L. Teng, and G. Wu, "Deep reinforcement learning for solving vehicle routing problems with backhauls," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 3, pp. 4779–4793, 2025.

[2] J. d. J. Cáceres Cruz, "Randomized algorithms for rich vehicle routing problems: From a specialized approach to a generic methodology," 2013.

[3] D. Bertsimas, "The probabilistic vehicle routing problem," 1988.

[4] J. Li, F. Wang, and Y. He, "Electric vehicle routing problem with battery swapping considering energy consumption and carbon emissions," *Sustainability*, vol. 12, no. 24, p. 10537, 2020.

[5] T. Vidal, R. Martinelli, T. A. Pham, and M. H. Hà, "Arc routing with time-dependent travel times and paths," *Transportation Science*, vol. 55, no. 3, pp. 706–724, 2021.

[6] R. Kumar, "The a* algorithm: A complete guide," Nov 2024.